C. REMARKS

1. Status of the Claims

Claims 1-32 are currently present in the Application and stand rejected. No claims have been added, cancelled, or amended.

2. Examiner Interview and Summary of the Present Invention

Applicants note with appreciation the interview conducted between the Examiner and Applicants' undersigned attorney on December 22, 2003. During the interview, Applicants' attorney explained Applicants' invention to the Examiner and discussed why the art of record did not teach or suggest Applicants' claimed invention.

Applicants' invention is a method / system / program product that manages a shared resource by determining whether a process that is requesting the resource is requesting read access or write access to the resource. The process allows a single read requestor to write to the resource or allows multiple readers to read from the resource simultaneously. Some additional dependent claims were also discussed and explained by Applicants' attorney. For instance, claim 5 claims that when a resource lock that controls the shared resource is released, the first process that requests the resource is granted the resource lock regardless of whether the process that acquires the lock had been waiting on the lock and regardless of whether other processes are waiting for the lock. Applicants' attorney described this activity as "line cutting" where a process that has not been waiting in a queue for the shared resource is able to immediately acquire the lock (and the resource), if no other process currently holds the lock. In this manner, the overhead involved with queuing the new process that is requesting the lock can be avoided as well as the overhead involved with putting the new requestor to sleep and overhead involved with waking up one of the waiting processes.

Two other dependent claims that were discussed in some detail were claims 8 and 9 which claim "speeding up" of one or more of the read requestors that acquire the lock (claim 8) and granting a "temporary time slice exemption" as one way of speeding up the read requestors (claim 9). Applicants' attorney explained that when multiple read requestors acquire a lock in "read only" mode, a process waiting to write to the resource needs to wait until *all* of the readers are finished. This can be problematic if one or more of the readers has a low system priority and, therefore, is time sliced or otherwise takes longer in order to finish its read processing. One way to speed up these slower processes is by providing them with a temporary time slice exemption (claim 9) so that the low priority process will not be time sliced and will therefore be able to finish its read processing more expeditiously.

Two other claims that were discussed were claims 10 and 11. Claim 10 is a claim directed to identifying a processes that are "upgraders." Upgraders are defined as processes that are requesting to atomically upgrade a shared resource. This typically happens when a reader of a shared resource wants to write a new value to the resource based upon the value that was read so that the value being written is consistent with the value currently being read. In this instance, the "reader" becomes a "writer" without allowing any interleaving writers to alter the current value of the resource (see Figure 7 and pages 28-30 of the specification for further details). Claim 10 deals with identifying an upgrader in the queue and granting the upgrader write access to the shared resource, while Claim 11 (depending upon Claim 10), deals with speeding up the upgrader process by boosting the process' priority.

Examiner Anya and Applicants' attorney discussed the art cited in the Office Action. Applicants' attorney explained that Waddington et. al (U.S. Pat. 6,041,384), does not teach or suggest determining whether a requestor is a read or write requestor, as asserted in the Office Action. Figure 9 of Waddington, on which the Office Action substantially relies for this proposition, teaches using a "coordinator" process to modify a database table. The two "locks" shown in Waddington's Figure 9 are database locks, which are quite different from the operating system kernel-level resource locks described and claimed by Applicants. Applicants' attorney explained that Waddington teaches, at step 920, that the coordinator process acquires a "shared mode" lock on the table and then at step 925 the same coordinator process acquires "exclusive mode" locks on each "table partition" being modified. Importantly, the coordinator process is not determining whether other processes are "read" or "write" processes. Indeed, in Waddington, no other processes even exist until the coordinator process *creates* them at step 930. The coordinator process manages the created "worker" processes by scheduling partitions for the worker processes to modify. The worker processes do not have to acquire any locks to the database or the partition because the coordinator process already gathered all needed database locks. Step 945, specifically cited in the Office Action, is simply the step in which the coordinator process assigns idle worker processes database partitions to modify.

Applicants' attorney and Examiner Anya further discussed the Mukherjee et al. reference (U.S. Pat. 6,466,978). The Mukherjee reference was used in the Office Action for the proposition of speeding up processes that acquire locks as claimed by Applicants in claims 8-11. Applicants' attorney explained that Mukherjee teaches, in a multimedia application environment, that locks to files are acquired and released with no teaching as to speeding up the *processes*. During the interview, the Examiner pointed to a portion of Mukherjee in support of his argument: "The lock is freed after t_{exp} unless it is refreshed 120 Thus, if the client needs to do a long editing operation, the file has to be refreshed periodically." Applicants' attorney explained that "refreshing" a lock (i.e., giving a particular process more time to perform shared resource operations) has *nothing to do* with speeding up the process that acquires the lock. Applicants' attorney explained that nowhere in Mukherjee, Waddington, or Vahalia do any of the reference discuss *speeding up processes that acquire locks* as described and claimed by Applicants. In Mukherjee, a slow process is able to repeatedly request more and more time to a lock whenever its expiration time (t_{exp}) is almost up. Rather than increase system performance, this would instead tend to slow down system performance as the speed of slow processes is not increased and such slow processes are able to use shared resources for an extended period of time, thus preventing other processes from accessing those resources.

3. Formal Drawings

Applicants note that the Examiner did not indicate whether the formal drawings, filed with the Application, have been accepted by the Examiner and/or draftsperson. Applicants respectfully request such an indication in the next communication.

4. **Claim Rejections – 35 U.S.C. § 103 – Waddington in view of Vahalia**

Claims 1-7, 10-17, 20-28, 31, and 32 stand rejected as unpatentable under 35 U.S.C. § 103 over U.S. Pat. No. 6,041,384 to Waddington et al. (hereinafter "Waddington") in view of U.S. Pat. No. 6,275,953 to Vahalia et al. (hereinafter "Vahalia). Applicants respectfully traverse the rejections.

The Office Action asserts that Waddington teaches a method of managing resources that includes determining whether a process identifier included in a queue corresponds to a read requestor or to a write requestor (Office Action p. 2, citing Waddington Step 935, Col 10. Lines 18-61. Upon review, however, Waddington is not teaching what the Office Action purports.

A copy of Waddington's Figure 9 appears below. Figure 9 includes Step 935 and describes the processing of a "coordinator process."

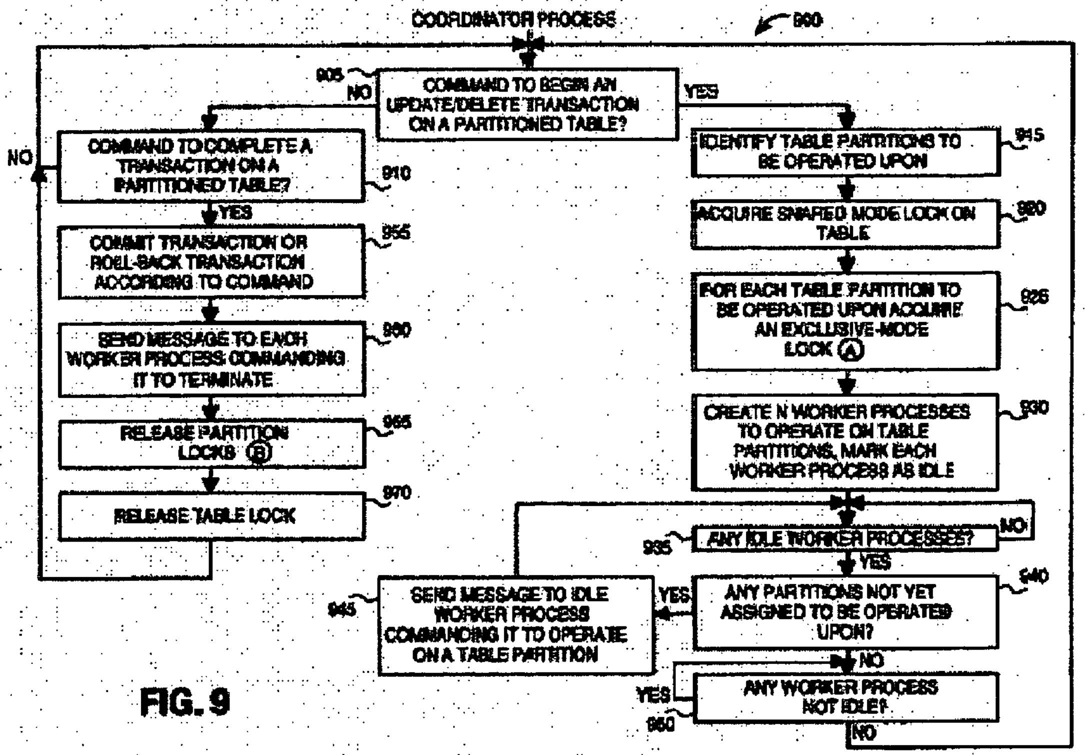

COORDINATOR PROCESS
900
905
COMMAND TO BEGIN AN UPDATE/DELETE TRANSACTION ON A PARTITIONED TABLE?
NO
YES
COMMAND TO COMPLETE A TRANSACTION ON A PARTITIONED TABLE?
910
NO
YES
COMMIT TRANSACTION OR ROLL-BACK TRANSACTION ACCORDING TO COMMAND
955
SEND MESSAGE TO EACH WORKER PROCESS COMMANDING IT TO TERMINATE
960
RELEASE PARTITION LOCKS B
965
RELEASE TABLE LOCK
970
IDENTIFY TABLE PARTITIONS TO BE OPERATED UPON
915
ACQUIRE SHARED MODE LOCK ON TABLE
920
FOR EACH TABLE PARTITION TO BE OPERATED UPON ACQUIRE AN EXCLUSIVE-MODE LOCK A
925
CREATE N WORKER PROCESSES TO OPERATE ON TABLE PARTITIONS, MARK EACH WORKER PROCESS AS IDLE
930
ANY IDLE WORKER PROCESSES?
935
NO
YES
ANY PARTITIONS NOT YET ASSIGNED TO BE OPERATED UPON?
940
YES
NO
SEND MESSAGE TO IDLE WORKER PROCESS COMMANDING IT TO OPERATE ON A TABLE PARTITION
945
ANY WORKER PROCESS NOT IDLE?
950
YES
NO

FIG. 9

In Figure 9 Wadding teaches that the "coordinator process," among other things, identifies partitions in a table to be operated upon (step 915), acquires shared mode lock on the entire table (step 920), and for each partition within the table to be operated upon, acquires an exclusive-mode lock (step 925). At step 930, Waddington teaches the creation of a number of worker processes that will perform the operations on the various table partitions. Step 935, cited in the Office Action, is actually a simple decision block that reads "Any idle worker processes?" and does not teach anything about determining whether a process is a read or write requestor. As discussed in the Examiner Interview, described at length in Section 2 above, Applicants' attorney pointed out to the Examiner that the "coordinator process" requested two different database locks without the coordinator process, or any other process, determining whether a process was a read or write requestor. Furthermore, Applicants claim two different types of requests (read and write) being made to the same resource. Waddington, meanwhile, teaches that a read request (shared-mode lock) is made to one resource (the database table, at step 920), while the write request (exclusive-mode lock) is made on a different resource altogether (the database partition, at step 925). Waddington, therefore, is not even teaching a method of managing a shared resource, instead Waddington is simply teaching that two different database locks are acquired before worker processes operate on table partitions.

The Office Action further contends that Figure 9 of Waddington teaches Applicants' step of "allowing the write requestor to write to the shared resource ... [and] allowing one or more successive read requestors to read from the shared resource...." citing Step 945 of Waddington as well as Col 10, lines 18-43 which describe the processing shown in Figure 9. As discussed during the Examiner interview, and as further explained below, Applicants completely disagree with this alleged teaching of Waddington.

Step 945 of Waddington simply assigns worker processes to operate on table partitions. In steps 920 and 925, the coordinator process acquired shared-mode locks of the database tables and then acquired exclusive-mode locks of the database partitions to be operated upon. In step 930, the coordinator process created a number of "worker processes" to actually perform the work, initially marking each worker process as "idle." In step 935, a determination is made as to whether there are any idle worker processes. If there are idle worker processes, step 935 branches to step 940 whereupon another determination is made as to whether there are any partitions that have not yet been assigned. If there are unassigned processes, step 940 branches to step 945 whereupon a message is sent to the idle worker processes to operate on a table partition. As previously explained, Waddington does not teach "read" and "write" processes. Instead, Waddington teaches a "coordinator process" that acts as a supervisor, or conductor, to acquire all the locks that will be needed (steps 920 and 925) and then database partitions to worker processes so that the worker processes, in a multiprocessing system, work on updating the various partitions simultaneously. In Waddington, the worker processes are not "read" or "write" requestors because they never request any locks. Instead, the coordinator process requests all of the locks and simply divvies the work to the various worker processes. When a worker process finishes operating on its assigned database partition, it once again becomes "idle" and the coordinator process can once again assign the worker process with another database partition on which to operate.

As is evident by the above discussion, the Office Action's reliance on Waddington is completely misplaced. Waddington does not teach or suggest, alone or in combination with Vahalia or Mukherjee, determining whether a process is a read or a write requestor as the coordinator process simply requests the locks without any process determining whether the coordinator process, or any process for that matter, is a read or write requestor. Moreover, the "read" and "write" locks requested by Waddington's coordinator process are to two different resources, not to one shared resource as claimed by Applicants. Finally, Waddington does not teach allowing a write requestor to write to the resource and allowing a read requestor to read from the resource. Instead, the worker processes are created after the locks are acquired by the coordinator process and each of the worker process writes to a the resources held by the coordinator process in "exclusive-mode." Waddington does not teach allowing multiple processes to read from the resource, as in Step 945, the worker processes are taught as "operating on" (i.e., writing to) the database partitions with no mention of multiple processes ever sharing a resource. Indeed, the focus of Waddington is distributing work to multiple worker processes in order to update database partitions more quickly. Having multiple processes trying to operate on a single database table or partition, as suggested by the Office Action, would thwart Waddington's stated intentions and slow the updating of the database partitions.

Claim 2 was also rejected with the Office Action primarily relying upon Waddington. This reliance is misplaced similarly to the Office Action's reliance on Waddington regarding claim 1. Claim 2 adds limitations of setting a resource lock as available, then setting the lock in "read mode" in response to the first process requesting to acquire a read lock to the resource, and then allowing other processes that are requesting a "read mode" lock to the resource to access the resource. Steps 920 and 925, as described above, only involve a single process (the coordinator process) acquiring "shared mode" (i.e., read mode) access to a database table (step 920), followed by acquiring "exclusive mode" (i.e., write mode) access to database partitions. These steps fail to teach or suggest Applicants' claimed invention set forth in claim 2 as they do not teach or suggest having multiple processes acquire a read lock on a resource. The Office Action further cites steps 1210 and 1220 in Waddington as teaching "setting the resource lock in a read mode in response to the first ... requestor accessing the available resource lock." Steps 1210 and 1215 of Waddington, detailed in Figure 12, teach processing performed by a worker process. Specifically, when a worker receives a command to operate on a table partition, the worker marks itself as "busy" (step 1210) and then acquires a "partition wait lock" in exclusive-mode (i.e., in "write mode") at step 1215. This teaching from Waddington teaches a worker process acquiring a "write mode" lock, as clearly shown in step 1215, which is simply not a read mode lock as claimed in Applicants' claim 2.

The rejection of Claim 3 cites the same sections of Waddington. Specifically, the Office Action asserts that Waddington teaches granting lock access to a first group of read requestors, citing elements from Figure 12 and corresponding text from Waddington's specification. However, as pointed out above, Figure 12 deals with "worker processes" that are assigned by the coordinator process in Figure 9 to "operate" on a table partition. In order to operate on table partitions, the worker process needs a "write-mode" (or "exclusive-mode") lock, which is performed at step 1215. No where in Figure 12 or the cited text does Waddington teach or suggest having multiple readers sharing a common lock in order to read from a resource.

The rejection of Claim 4 cites the same general sections of Waddington as were used in the rejections of Claims 2 and 3. Again, Waddington is teaching the creation of multiple worker processes that "operate" on a database table partition. When the worker process "...ceases to be an idle worker..." it simply means that the worker process has been assigned a partition, by the coordinator process, on which to operate. As no read locks are acquired (the worker needs an exclusive-mode lock on the partition in order to operate), the worker never sets a "woken up flag for each _read requestor_" (because the workers _are not_ read requestors in the first place as explained in the traversal of the rejections of Claims 2 and 3).

Applicants' claim 5 includes the limitations of releasing the resource lock and granting the lock to the first process that requests the lock after the lock is released. In essence, this claim is claiming a situation wherein a new process requests the lock after the lock is released and without waiting in a queue for the lock. As discussed in the Examiner interview, this is the "line cutting" situation where a process gets the resource lock even though other processes are waiting for the lock in a queue. This "line cutting" ability is further clarified in Claim 6 that includes the limitation that the process that receives the lock is not listed in the queue. Figure 9 of Waddington, cited in the Office Action's rejection of these claims, does not teach or suggest these claim limitations. The steps cited in Waddington (steps 960, 965, and 970), are performed when the coordinator process receives a command that does not involve beginning an update or delete database transaction (see step 905 branching to step 910). When this occurs, the coordinator process halts any worker process that is currently operating on the given database partition (step 960) by sending it a message, and the locks to the partition and table are released (steps 965 and 970, respectively). In these steps, Waddington does not teach or suggest the mechanism for granting the locks to a next requestor. Indeed, due to the database processing being performed by Waddington, it is more likely that a specific process, waiting for the lock, is granted the lock rather than granting the lock to any process that happens to request the lock after the lock has been released.

The rejections of Claims 6 and 7 is traversed because the Office Action admits that Waddington does not teach that lock requestors are stored in a queue (see Office Action, page 2). This was the Office Action's stated reason for including the Vahalia reference, however the Vahalia reference was not used to show a queue being used in the manner claimed in Applicants' claims 6 and 7.

Claims 10 and 11 include the limitations of identifying an "upgrader" process in the queue (claim 10), and boosting the upgrader process' priority. The Office Action asserts that

Vahalia teaches these limitations, referencing Col. 20, lines 60-67. The referenced section of Vahalia is reproduced below:

> In step 186, execution branches to step 187 if the file is locked. In step 187, execution branches to step 188 to open the file for the client's access and to grant a lock if the client should be granted a lock to open the file for the client's access. For example, if the client has a read lock on the file and would like to perform a read-write operation on the file, then the client's read lock may be promoted to a write lock so long as other clients do not also have a read lock on the file and so long as a higher priority client is not waiting for a write lock on the file. If the client should not be granted a lock to open the file for the client's desired access, then execution continues from step 187 to step 189. In step 189, the client is placed on a wait list for the file.

The referenced text describes processing of an "upgrader" process which is shown in Vahalia's Figure 17. Importantly, while Vahalia is describing an upgrader process, Vahalia does not teach or suggest identifying the upgrader from a queue. Figure 17 of Vahalia shows the steps involved in a "file access module" in which a client upgrades its access to a file from a read-only access to a write access. In addition, Applicants' claim 11 includes the limitation of "boosting" the priority of the upgrader process. No only does Vahalia not teach this limitation, he actually teaches away from it. Specifically, as cited above, Vahalia allows promotion to a write lock "so long as a higher priority client" is not waiting for a write lock on the file." (Vahalia, Col 21, lines 1-2).

Claims 2-7, 10, and 11 each depend, either directly or indirectly, on Claim 1, and are therefore allowable because, as explained above, Claim 1 is allowable. In addition, these claims are allowable for the reasons set forth above. Claim 12 is a information handling system claim that includes the same limitations as Claim 1, so is allowable for at least the same reasons as Claim 1 is allowable. Claims 13-17, 20, and 21 each depend, either directly or indirectly, on Claim 12, and are therefore allowable because Claim 12 is allowable. Claim 22 is a computer program product claim that includes the same limitations as Claim 1, so is allowable for at least the same reasons as Claim 1 is allowable. Claims 23-28, 31, and 32 each depend, either directly or indirectly, on Claim 22, and are therefore allowable because Claim 22 is allowable. As set forth above, the rejections of Claims 1-7, 10-17, 20-28, 31, and 32 have been traversed.

5. <u>Claim Rejections – 35 U.S.C. § 103 – Waddington in view of Vahalia in further view of Mukherjee</u>

Claims 8, 9, 18, 19, 29, and 30 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Waddington, Vahalia, and U.S. Patent No. 6,466,978 to Mukherjee et al. (hereinafter "Mukherjee"). Applicants respectfully traverse the rejections. As an initial matter, each of the rejected claims depends directly or indirectly on an allowable independent claim, as explained in Section 4 above, and are therefore allowable for at least the same reasons as the independent claims are allowable. Notwithstanding the claims allowability on these grounds, claims 8, 9, 18, 19, 29, and 30 are also allowable for the reasons set forth below.

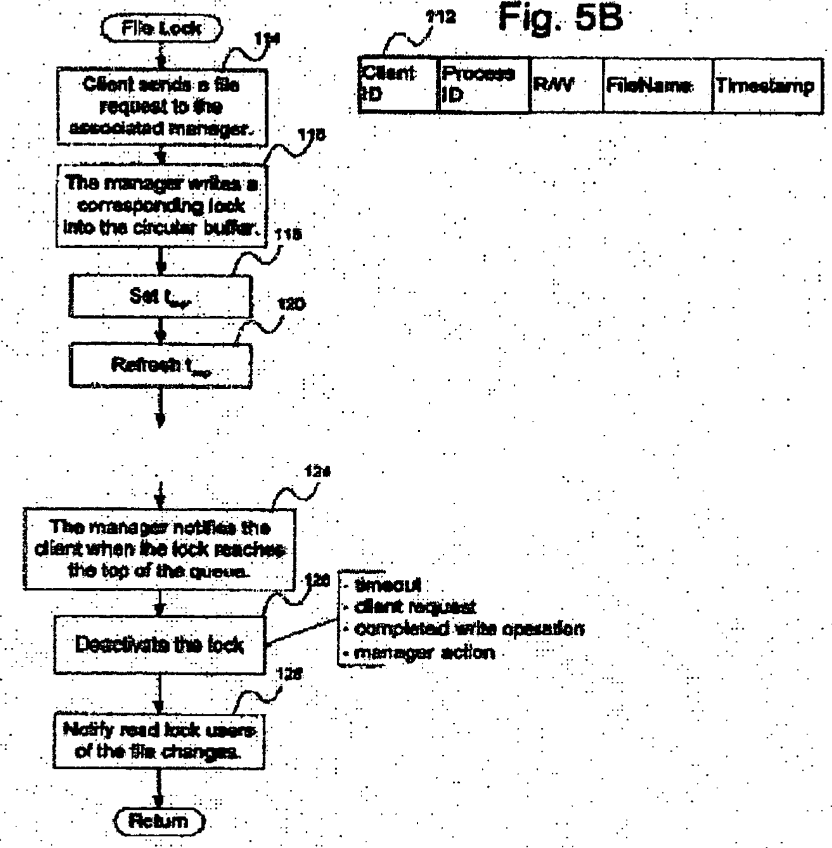
Claims 8, 9, 18, 19, 29, and 30 all deal with speeding up a process while it is holding a lock. Claims 8, 18, and 29 each claim "speeding up processing for one or more read requestors that acquire the resource lock." Claims 9, 19, and 30 depend upon claims 8, 18, and 29, respectively, and add the limitation that the "speeding up includes granting ... a temporary time slice exemption."

In rejecting these claims, the Office Action cites Mukherjee as "teach[ing] speeding up processing that includes granting one or more read requestors a temporary time slice exemption," citing Mukherjee's teaching of a "Refresh 120" which is described on col. 8, line 49 – col. 9, line 17. Upon closer inspection, however, it is evident that Mukherjee is not teaching what the Office Action posits.

As discussed during the Examiner interview, Mukherjee teaches aspects of handling multimedia applications where locks to files are acquired and released. Mukherjee does not teach or suggest ways in speeding up processes that acquire such file locks. During the Examiner interview, the Examiner pointed to a portion of Mukherjee in support of the rejection: "The lock is freed after t_{exp} unless it is refreshed 120 Thus, if the client needs to do a long editing operation, the file has to be refreshed periodically." (col. 9, lines 2-3) Applicants' attorney explained that "refreshing" a lock (i.e., giving a particular process more time to perform shared resource operations) has <u>*nothing to do*</u> with speeding up the process that acquires the lock. The cited section of Mukherjee describes the processing that is shown on Figures 5A and 5B, which has been reproduced below:

Fig. 5A

Fig. 5B


File Lock
114
Client sends a file request to the associated manager.
116
The manager writes a corresponding lock into the circular buffer.
118
Set t_exp
120
Refresh t_exp
124
The manager notifies the client when the lock reaches the top of the queue.
126
- timeout
- client request
- completed write operation
- manager action
Deactivate the lock
128
Notify read lock users of the file changes.
Return
112
Client ID
Process ID
R/W
FileName
Timestamp

As is evident by looking at Figure 5A and 5B along with the text, Mukherjee describes setting an expiration time (step 118) and, if needed refreshing the timer (step 120). This is done so that the process does not have the lock taken away before it is finished using the associated resource. The lock manager is shown deactivating the lock in step 126, which is shown occurring when either (1) the expiration time has elapsed (a "timeout" condition occurred), (2) the client process requests deactivation, (3) the write operation has completed, or (4) the lock manager acts upon the lock.

It is important to note that when a process requests a "refresh" of the lock, the process receives additional time, but the process is not sped up, nor does Mukherjee teach or suggest granting the process a temporary time slice exemption in order for the process to perform its work more quickly. In Mukherjee, the operating system kernel is free to swap out the process holding the lock as its speed has not been boosted and it has not received a temporary time slice exemption. As set forth above, the rejections of claims 8, 9, 18, 19, 29, and 30 has been rejected because Mukherjee, alone or in combination with Waddington and Vahalia does not teach or suggest the process speed boosting steps set forth in these claims.

Conclusion

As a result of the foregoing, it is asserted by Applicants that the remaining claims in the Application are in condition for allowance, and Applicants respectfully request an early allowance of such claims.

Applicants respectfully request that the Examiner contact the Applicants' attorney listed below if the Examiner believes that such a discussion would be helpful in resolving any remaining questions or issues related to this Application.

Respectfully submitted,

By [illegible]
Joseph T. Van Leeuwen
Attorney for Applicant
Registration No. 44,383
Telephone: (512) 301-6738
Facsimile: (512) 301-6742